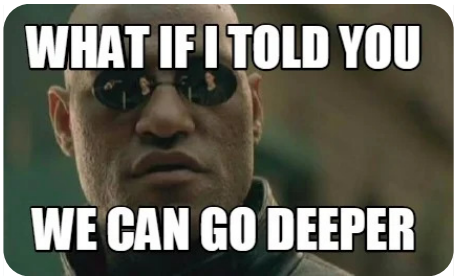
{EPITECH}

JAVA SEMINAR_

< DAY 04 - INHERITANCE />

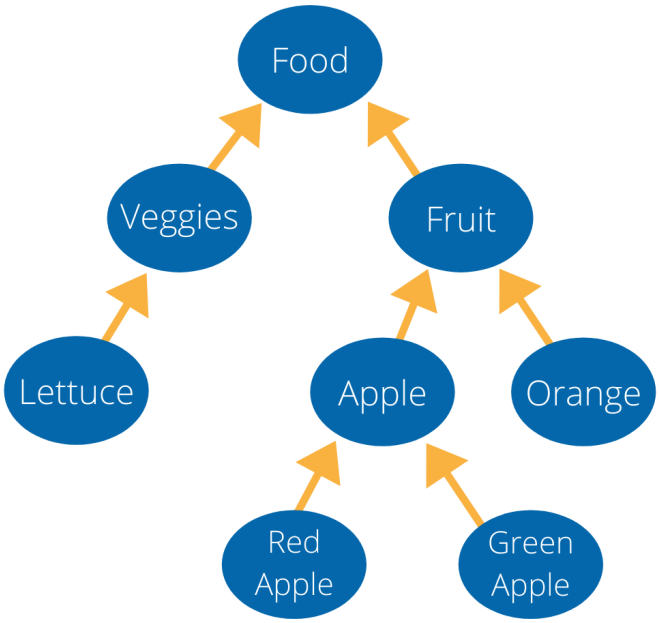
JAVA SEMINAR



Let's dig deeper into OOP... Today, you will discover:

- ✓ enum
- ✓ static keyword with methods (also called ”static method”)
- ✓ protected visibility
- ✓ inheritance

The last one, inheritance, is probably the most important notion in OOP:
For instance, an `Apple` class inherits all the non private methods and attributes from its parent class `Fruit`, such as an `eat` method and a `color` attribute. It can also have its own method, like `peel`.



Because inheritance is limitless, the `GreenApple` class inherits from the `Apple` class and from the `Fruit` class all the non private methods and attributes.

Unless specified otherwise, all messages must be followed by a newline and the names of the getter and setter for `Attribute` will always be like `getAttribute` and `setAttribute`.
For instance, attribute `Bobby` will have `getBobby` and `setBobby`.

FYI, this name convention is known as *CamelCase*.

Exercise 01

Delivery: `./Animal.java`

Create a new `Animal` class.
It has a protected enum `Type` with 3 possible values: **MAMMAL**, **FISH** and **BIRD**.
These variables will be used to pass your Animal's type to the protected constructor.
This constructor takes 3 mandatory parameters:

- ✓ the name of your animal ;
- ✓ its number of legs ;
- ✓ its type, among the 3 previously-created attributes.

Store the parameters inside new attributes named `name`, `legs` and `type`.
Create getters for each of these attributes.

Even though accidents can happen, we are generous gods (well, at least, I am!) and will consider that an animal will never lose a leg after creation. That's why there is no setter for the `legs` attribute.
For the other attributes, think about it by yourself. There are some logical reasons for it.

> ⚠ Be careful, the getter for `type` doesn't return the enum value directly. (check the example)

Last, but not least, during its creation your Animal must say `My name is [name] and I am a [type]!`

> ⚠ The constructor is *protected* and not *public*, thus there may be cases when you can't instantiate the object directly.
> Read the documentation for more information.

```java
public class Example {
    public static void main(String[] args) {
        Animal isidore = new Animal("Isidore", 4, Animal.Type.MAMMAL);

        System.out.println(isidore.getName() + " has " + isidore.getLegs() + " legs and is
            a " + isidore.getType() + ".");
    }
}
```

Terminal

```
$> java Example
My name is Isidore and I am a mammal!
Isidore has 4 legs and is a mammal.
```

Exercise 02

Delivery: `./Animal.java`

> In this exercise, use plural when necessary.
> **Autograder says:** The plural of "fish" is "fish".
> **Autograder says (bis repetita):** If you see "there are" after a "0", it means you must use plural nouns (see here)

Implement a private static field `numberOfAnimals` and it's getter `getNumberOfAnimals` which:

- ✓ returns the number of `Animal` instances ;
- ✓ displays a sentence such as one of the following:
  - If the number of animals is 2 or more : `There are currently [x] animals in our world.`
  - If the number of animals is exactly 1 : `There is currently 1 animal in our world.`
  - If the number of animals is exactly 0 : `There are currently 0 animals in our world.`

The method to handle this is up to you.

You must also implement 3 variants of this for the following private static fields:

- ✓ `numberOfMammals` ;
- ✓ `numberOfFish` ;
- ✓ `numberOfBirds`.

They return the number of instance for each type of instance / alive animals in the world.
They display a message such as `There (is|are)currently [x] [type](s)in our world.`

An example would be `There are currently 3 mammals in our world.`.

> Animals never die.

Exercise 03

Delivery: `./Animal.java`, `./Cat.java`

Create a new `Cat` class that inherits from `Animal`.
It has a private `color` attribute, with a getter only (a setter would be animal cruelty!).
When `Cat` is created, display `[name]: MEEEOOWWWW`

The name of the cat should always be specified as the first parameter of its constructor.
However, the color can be specified as its second parameter, but is not mandatory.
Its `legs` must be set to 4 and its `type` to MAMMAL by default.

If no color has been specified during creation, the default color will be grey.

In Java a constructor can call a constructor from the upper class. Use it!

Add a public `meow` method to your `Cat` class that:

- ✓ does not take any parameters ;
- ✓ displays `[name] the [color] kitty is meowing.` when calling it.

```java
public class Example {
    public static void main(String[] args) {
        Cat isidore = new Cat("Isidore", "orange");

        System.out.println(isidore.getName() + " has " + isidore.getLegs() + " legs and is
            a " + isidore.getType() + ".");
        isidore.meow();
    }
}
```

Terminal

```
$> java Example
My name is Isidore and I am a mammal!
Isidore: MEEEOOWWWW
Isidore has 4 legs and is a mammal.
Isidore the orange kitty is meowing.
```

Exercise 04

Delivery: `./Animal.java`, `./Cat.java`, `./Shark.java`, `./Canary.java`

Create `Shark` and `Canary` class, which both inherit from `Animal`. It receives its name as parameter during construction and displays `A KILLER IS BORN!` during creation.
Its `legs` should be set to 0 and its `type` to FISH.
The `Shark` class must also have a private `frenzy` attribute, `false` by default.
The `Shark` class has a `smellBlood` method to it that:

- ✓ takes a Boolean as parameter ;
- ✓ returns nothing ;
- ✓ changes the value of `frenzy` to the value passed as parameter.

Finally, add a `status` method displaying one of the two following messages:

- ✓ `[name] is smelling blood and wants to kill.`, if `frenzy` is true ;
- ✓ `[name] is swimming peacefully.`, if `frenzy` is false.

Your `Canary` class must have a private `eggs` attribute, indicating how many eggs it has laid in its life.
During initialization, `Canary` only takes one parameter: its name.
Initialize `legs` to 2, `type` to BIRD and `eggs` to 0.
Display `Yellow and smart? Here I am!` when a `Canary` is created.
Add a `getEggsCount` method that returns the number of eggs laid by the Canary.
Add a `layEgg` method that increases the number of eggs laid by 1.

```java
public class Example {
    public static void main(String[] args) {
        Canary titi = new Canary("Titi");
        Shark willy = new Shark("Willy"); //Yes Willy is a shark here !
        willy.status();
        willy.smellBlood(true);
        willy.status();
        titi.layEgg();
        System.out.println(titi.getEggsCount());
    }
}
```

Terminal

```
$> java Example
My name is Titi and I am a bird!
Yellow and smart? Here I am!
My name is Willy and I am a fish!
A KILLER IS BORN!
Willy is swimming peacefully.
Willy is smelling blood and wants to kill.
1
```

Exercise 05

Delivery: `./Animal.java`, `./Cat.java`, `./Shark.java`, `./Canary.java`

Willy, your dear shark, needs to eat...
Add a public `canEat` method to your `Shark` class that:

- ✓ takes an `Animal` as parameter ;
- ✓ returns a boolean indicating wether or not the `Shark` can eat the `Animal`.

Add another public method `eat`, also taking an `Animal` as parameter.

When the method is called, display:

- ✓ `[Shark's name] ate a [Animal's type] named [Animal's name].` if the parameter can be eaten ;
- ✓ `[Shark's name]: It's not worth my time.` if the parameter can't be eaten ;

When `Shark` has eaten, its `frenzy` attribute must be set to false after calling this method.

> ⚠ Your shark cannot eat itself.

Exercise 06

Delivery: `./Animal.java`, `./Cat.java`, `./Shark.java`, `./Canary.java`, `./BlueShark.java`, `./GreatWhite.java`

Create two new classes, `BlueShark` and `GreatWhite`, which both inherit from the `Shark` class.
They both take their name during construction as a mandatory argument.

They are almost identical to the original `Shark` class, except that they are picky eaters.

`BlueShark` refuses to eat anything but FISH. For that, only the `canEat` should be modified.

`GreatWhite` refuses to eat `Canary`. Even worse, it will only say `[name]: Next time you try to give me that to eat, I 'll eat you instead.` if you try to feed it with a `Canary`.

If a `GreatWhite` eats another `Shark`, it says (after it's done eating) `[name]: The best meal one could wish for.`

Except for those differences, their `eat` method does the exact same thing as for the `Shark` Class.

> Cleverly use the `Override` annotation and the `instanceof` keyword.

v 3.4.1